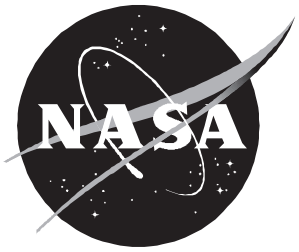
NASA Technical Paper 3429

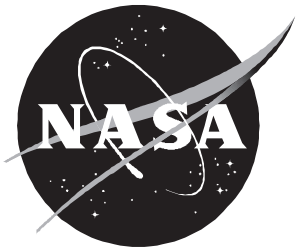


Parallel Grid Generation Algorithm for Distributed Memory Computers

Stuti Moitra and Anutosh Moitra

February 1994

NASA Technical Paper 3429



Parallel Grid Generation Algorithm for Distributed Memory Computers

Stuti Moitra
Langley Research Center • Hampton, Virginia

Anutosh Moitra
High Technology Corporation • Hampton, Virginia

National Aeronautics and Space Administration
Langley Research Center • Hampton, Virginia 23681-0001

February 1994

Symbols

$\mathbf{A}, \mathbf{B}$	vectors used in orthogonalization
C	analytical blending function
ds	specified grid spacings at boundary
E	variable analogous to homotopic parameter
f, h	homotopic mappings
H	boundary curve
I, J, K	loop indices
m	exponent used to distribute homotopic parameter
P	number of processors
p, q	exponents used for orthogonalization
r, s	exponents used to remove intersection of trajectories
S	speed-up ratio
T	execution time
U	unit interval
x, y, z	Cartesian coordinate directions
η	homotopic parameter
λ	scaling function
τ	circumferential parameter for defining boundary shapes

Subscripts:

i	inner boundary
o	outer boundary
p	number of processors

Abbreviations:

CAS	Computational Aerosciences Project
CFD	computational fluid dynamics
CFS	concurrent file system
I/O	input/output
iPSC	Intel parallel supercomputer
MIMD	multiple instruction multiple data
SCSI	small computer systems interface
SOR	successive overrelaxation method
SRM	system resource module

Summary

A careful examination of the algebraic scheme for grid generation based on homotopic relations has revealed highly parallelizable characteristics. Implementation of this parallel scheme on the Intel iPSC/860 computer has resulted in efficient software that demonstrates good correlation of speed-up ratios with the number of processors used. The software accepts discrete point data sets as input geometries. It also generates blended wing-body configurations by a semianalytic procedure. The Intel concurrent file system (CFS) is used for efficient implementation of parallel I/O operations. The number of points in each coordinate direction is normally chosen to be a multiple of the number of processors for perfect scalability; however, this is not a strict requirement. The algebraic procedure for grid generation is explicit and therefore requires minimal interprocessor communication overhead. The grid is generated in cross-sectional planes that are stacked along the longitudinal axis of the input geometry to produce a quasi-three-dimensional grid system. Multiple levels of parallelism are investigated in this report. In the first level of parallelism, the cross-sectional planes are distributed over the processors in an interleaved manner. The second level of parallelism is achieved by distributing the radial lines of a cross-sectional planar grid over the processors. The Express[1] system of software tools has been used to enhance portability of the grid-generation software. The Express version of the code has been implemented successfully on the Intel iPSC/860 computer as well as a network of Sun workstations. Test cases for the programming modes consist of a blended wing-body configuration in the case of the analytically generated input geometry and a high-speed civil-transport configuration for the discrete case. The analytical geometry is defined by 64 cross-sectional planes, each containing 32 points along the circumference. The discretely defined geometry contains 64 cross-sectional planes, each with 42 circumferential points on the surface. The number of grid points in the radial direction is 49 in both cases. All cases demonstrate the parallel behavior of the software in all programming modes and on all hardware platforms used.

Introduction

Grid generation is an indispensable prerequisite for computational fluid dynamics (CFD) research. Given the importance and computationally intensive nature of the grid-generation process, evidence of parallel grid-generation methods in existing literature is surprisingly scant.

[1] Express is a registered trademark of ParaSoft Corporation.

Current grid-generation methods are essentially sequential. These methods generally compute near-orthogonal trajectories that span the gap between specified inner and outer boundaries. Computation of each point on a trajectory is usually heavily dependent on the coordinates of the neighboring points. The dependence is larger for grid generators employing systems of partial differential equations than for those generators based on algebraic interpolation schemes; nevertheless, parallelization of the scheme in either case generally involves extensive communication between processors to transfer relevant information about surrounding points.

In view of the CFD issues outlined by Holst et al. (ref. 1), development of advanced parallel algorithms for generating grids for computation of flows about aerospace vehicles is a major requirement. The advent of high-performance parallel computers has already prompted substantial research in the realm of parallel flow solvers. In comparison, the development of parallel grid-generation methods is lagging behind. Future time-dependent multidisciplinary applications in the NASA Computational Aerosciences Project (CAS) require integrating a regridding step into the CFD simulation software in order to reflect changes in aircraft location and aerodynamic control surface positions. This goal can only be achieved by developing parallel grid-generation algorithms that can be coupled with parallel flow solvers for execution on massively parallel computers.

One parallel scheme for grid generation consists of decomposing the computational domain into as many parts as there are processors in the hardware and distributing the computational work for these parts as evenly as possible among the processors. A broad overview of the desirable characteristics of parallel grid-generation systems has been given by Gentzsch (ref. 2). The levels of performance of these systems are mainly determined by four factors: (1) the degree of parallelism in the algorithm, (2) the evenness of computational load balancing, (3) the amount of interprocessor communication necessary, and (4) correlation of performance with the number of processors. In the ideal system, these requirements are in perfect harmony. Most currently available serial or vector algorithms are not optimally suited for achieving this harmony. These existing algorithms were developed for scalar computers and are often difficult to parallelize because of their sheer complexity, (e.g., their implicitness, nonlinearity, and recursion). It is imperative that future research be directed toward development of special parallel algorithms that facilitate near-optimal mapping of advanced hardware and topology. This goal can be met by simple explicit

algorithms that minimize interprocessor communication. Parallel implementation of such methods entails little more than distributing the main loop index over the processors.

A parallel grid-generation scheme must address several key issues unique to problems of computational geometry and grid generation. Generally, a structured grid must be body conforming, smoothly varying, properly clustered, and nearly orthogonal at relevant boundaries. In addition, the grid must not be discontinuous or overly skewed, and it should not waste points in regions of the domain where little change in the relevant physical properties takes place. The above considerations make it clear that grid generation often necessitates a global knowledge of the domain to be generated. However, this global concept is difficult to maintain when the domain is decomposed and distributed to individual processors and constitutes a major obstacle in developing truly parallel grid-generation algorithms.

Two implementations of a structured grid-generation algorithm that solve partial differential governing equations by a successive overrelaxation (SOR) method have been reported by Gentzsch and Häuser (ref. 2). The first implementation on the Alliant FX/80 shared memory computer is based on dividing the computational domain into smaller portions, which are simultaneously processed on individual processors. Each first and last inner grid line of a partition is a boundary grid line of the neighboring partitions. After each iteration of the SOR algorithm, the boundary data are exchanged among neighboring grids through interprocessor communication. A synchronization point is needed at the end of every iteration to ensure that all segments begin the next iteration with updated boundary data. The second implementation is for the distributed memory, tree-structured system TX3 of iP-Systems. In this model, the problem is successively divided into two subtasks of identical complexity. Because the partitioning of the complex physical domain is highly unstructured, the organization of the communication pattern between the processors is a complex task.

A parallel algorithm for automatic mesh refinement has been presented by Berger (ref. 3). This scheme uses nested grids with recursively refined mesh spacings in regions where greater resolution is needed in the solution. The finer grids are superimposed on the underlying coarser grids. A binary decomposition technique is used to partition the domain so that the workload is distributed as evenly as possible. An a priori estimate of the computational work over the entire domain is essential for applying this technique. The partition is a function of time because a repartitioning is necessary whenever the grid hierarchy is changed.

Löhner et al. (ref. 4) have reported an interesting algorithm for parallel unstructured triangular grids. This algorithm is based on an advancing front technique for filling empty space that introduces points with distribution prescribed by background grids. A crucial condition for achieving parallelism is that the neighborhoods of the points to be introduced do not overlap. Therefore, distance enables parallelism. Unstructured grids are free from the constraints of orthogonality and continuity of grid lines. However, smoothness of variations in shape and size of grid cells is still a constraint. A Laplace smoother is used in each subdomain to improve the uniformity of the grid. A communication scheme for exchange of information between processors allows movement of the boundary points at subdomain interfaces. This algorithm has been implemented with the host-node programming model on the Intel hypercube.

The algorithm presented in this paper employs an algebraic homotopic procedure that develops into a completely explicit numeric scheme. (A homotopy is essentially a family of maps of smooth lines between two given surfaces.) The maps are generated by the smooth variation of a parameter in a unit interval between the given surfaces. The resulting procedure is inherently parallel. This algorithm has been implemented on the Intel iPSC/860 hypercube machine and a network of Sun workstations. A sequential version of the algorithm (refs. 5 and 6) has been used for high-speed flow simulation for aircraft over the past several years. The basic method, which is completely explicit, allows arbitrary decomposition of the domain into blocks that can be processed on individual processors in parallel. The software has a built-in provision for analytically generating blended wing-body surface geometries, but it also accepts input geometries specified as a set of discrete points. The grid generation scheme uses a homotopic blending technique for generating surfaces between the body surface and a specified outer boundary surface while maintaining near-orthogonal trajectories in the vicinity of the body surface. The grid is generated in cross-sectional planes along the longitudinal axis of the input geometry. For the analytically generated input geometry, the schemes for generating the body surface and the grid points can be integrated into one procedure that allows the computation of each grid point independently of other grid points. Any required boundary data can be generated in each individual processor, thus virtually eliminating the need for expensive interprocessor communications. In the

discrete case, necessary boundary data can be read in parallel into each processor from the CFS.

Theory and Mathematical Development

Grid Generation

The grid is generated through determination of a family of curves representing a smooth and gradual transition from the given inner boundary (x_i, y_i) to the outer boundary (x_o, y_o) in two-dimensional planes at each z-station. Assuming that the body surface coordinates are available, either as an analytic description or as a set of discrete points, a distribution of a homotopic parameter η is specified. The η distribution may be specified by means of polynomials, exponents, or trigonometric functions while ensuring that $\eta = 0$ on the body surface and $\eta = 1$ on the outer boundary. A shape transition function C for the grid is then specified by

$$C(\eta) = 1 - \eta^m \tag{1}$$

where m is a positive exponent providing control over line spacing near boundaries. Thus $C = 1$ on the inner boundary and $C = 0$ on the outer boundary, with a smoothly varying distribution between boundaries. The function C is taken to be independent of z; it retains the same value at each z-station. Essentially the transition curves are defined by a family of maps given by the homotopy

$$\{h_\eta : \mathbf{A} \to \mathbf{B} | \eta \varepsilon U\} \tag{2}$$

where U is the unit interval $[0, 1]$. The inner and outer boundaries H_i and H_o are two homotopic maps such that $h_o = H_i$ and $h_1 = H_o$. A scaling function λ for the grid lines is then defined such that $\lambda_i < \lambda < \lambda_o$, where λ_i and λ_o are the scaling function values associated with the inner and outer boundaries. Since the size of the inner surface and possibly that of the outer surface vary with z, λ is not independent of z. The grid is defined at each z-station in terms of the functions $C(\eta)$ and $\lambda(\eta)$, and a natural correspondence is established between the grid points at the various z-stations. Coordinates of each boundary are expressed in terms of a parameter τ. For the inner boundary

$$\left.\begin{aligned} x_i &= x_i(\tau) \\ y_i &= y_i(\tau) \end{aligned}\right\} \tag{3}$$

and similar expressions denote the coordinates of the outer boundary. In polar coordinates, τ could represent the angular coordinate. If y can be expressed as a single-valued function of x, then $\tau = x$. For shapes of greater complexity, the choice of variables varies. An arithmetic averaging between the boundaries yields the simplest family of grid lines given by

$$\left.\begin{aligned} x(\eta, \tau) &= \lambda\{C(\eta)\, x_i(\tau) + [1 - C(\eta)]\, x_o(\tau)\} \\ y(\eta, \tau) &= \lambda\{C(\eta)\, y_i(\tau) + [1 - C(\eta)]\, y_o(\tau)\} \end{aligned}\right\} \tag{4}$$

where (x_i, y_i) and (x_o, y_o) are corresponding points on the inner and outer boundaries, as shown in figure 1. Geometric averaging may be used for smoother transition in cases involving complex geometries with sharp corners.

Smooth variations of the inner and outer boundaries with respect to z results in smoothly varying grid lines in the z-direction. Particular constant values of the η-parameter generate specific curves of the family described by equations (4), and a prechosen distribution of η determines the spacing of the resulting set of curves in each cross-sectional plane. Judicious modifications of the function $C(\eta)$ and the distribution of η result in approximate orthogonality of grid lines at physical boundaries and concentration of grid lines near boundaries. The planar grids are stacked along the Z-axis to produce a three-dimensional grid system, as shown in figure 2.

Orthogonality and Spacing Control

Orthogonality and prescribed spacing near physical boundaries are two important characteristics of grid systems used in CFD studies. The grid must also be smooth and free from intersecting grid lines of the same family. The present method provides orthogonality and control over spacing while maintaining smoothness and preventing grid intersections by a technique for local perturbation of the homotopic parameter. The required amount of perturbation is derived from the boundary data exploiting the stability properties of homotopic maps under perturbation. A property is said to be stable if wherever $f_o : x \to y$ possesses the property and $f_t : x \to y$ is a homotopy of f_o, then for some $\varepsilon > 0$, each f_t with $t < \varepsilon$ also possesses the property. The properties of smoothness of the grid and conformity of the overall grid with the given boundaries are stable under slight deformations of the map caused by small perturbations of the homotopic parameter. The basic interpolation scheme, equations (4), may be written in a modified form as

$$\left.\begin{aligned} x &= x_i E^p + x_o(1 - E^p) \\ y &= y_i E^q + y_o(1 - E^q) \end{aligned}\right\} \tag{5}$$

where

$$E = 1 - \eta^m \tag{6}$$

The subscripts i and o denote the inner and outer boundaries. Here E is analogous to a homotopic parameter. Modifications of E, therefore, cause slight deformations of a given map and may be used to achieve orthogonality and prescribed spacing at the inner boundary. This control is provided through the use of the exponents p and q. These exponents are not constants and their values must be determined from the boundary data subject to the constraints of orthogonality and required spacing.

The orthogonality condition requires that the vectors **A** and **B** in figure 3 be orthogonal. The vector **A** is found by connecting the point (x_i, y_i) on the inner boundary and the point (x, y) lying just off the boundary on the trajectory in question. The second vector **B** passes through the point (x_i, y_i) and a point (x', y') on the line passing through (x_i, y_i) and parallel to the line joining (x_{i+1}, y_{i+1}) and (x_{i-1}, y_{i-1}) on the inner boundary. The orthogonality condition is satisfied if the dot product of the vectors **A** and **B** is zero, that is,

$$\mathbf{A} \cdot \mathbf{B} = 0 \tag{7}$$

which translates to

$$(x - x_i)(x' - x_i) + (y - y_i)(y' - y_i) = 0 \tag{8}$$

Substituting equations (5) into equation (8) one obtains

$$(x_o - x_i)(1 - E^p)(x' - x_i) + (y_o - y_i)(1 - E^q)(y' - y_i) = 0 \tag{9}$$

The second condition, that of specified spacing ds in figure 3, can be written as

$$(x - x_i)^2 + (y - y_i)^2 = ds^2 \tag{10}$$

Substitution of equations (5) into equation (10) results in

$$[(x_o - x_i)(1 - E^p)]^2 + [(y_o - y_i)(1 - E^q)]^2 = ds^2 \tag{11}$$

The exponents p and q can be solved from equations (9) and (11) and are given by

$$p = \frac{\ln\{1 + \mathbf{B}(y' - y_i)/[(x_o - x_i)(x' - y_i)]\}}{\ln E} \tag{12}$$

and

$$q = \frac{\ln\{1 - [\mathbf{B}/(y_o - y_i)]\}}{\ln E} \tag{13}$$

where

$$\mathbf{B} = \frac{ds}{\sqrt{1 + (y' - y_i)^2/(x' - x_i)^2}} \tag{14}$$

and E has the value corresponding to the homotopic curve lying next to the inner boundary. The values of p and q given by equations (12) and (13) will result in constant spacing ds between the first homotopic curve and the boundary as well as near orthogonality between the trajectory emanating from (x_i, y_i) and the inner boundary.

Strict imposition of orthogonality in regions of high boundary curvature often results in intersection of the trajectories. Intersecting trajectories can be separated through modification of p and q by the use of further exponents r and s, such that

$$\left.\begin{aligned} x &= x_i E^{p^r} + x_o(1 - E^{p^r}) \qquad (r < 1.0) \\ y &= y_i E^{q^s} + y_o(1 - E^{q^s}) \qquad (s < 1.0) \end{aligned}\right\} \tag{15}$$

Using constant values for r and s, however, reduces orthogonality. In order to maintain orthogonality near the boundary, r and s are made to decay as one proceeds along trajectories outward from the inner boundary. This decay is achieved by making r and s functions of E such that r and $s = 1$ at the inner boundary and r and $s = 0$ at the outer boundary.

Surface Geometry Definition

As mentioned before, the input surface geometry may be defined either analytically or discretely. In the analytic case any explicit analytic formula may be used. A semianalytic method for defining blended wing-body configurations has been described in reference 5. The analytic expressions for surface definition can be integrated into the grid-generation scheme to produce one set of governing equations for defining any grid point. The exact form of these analytic expressions is of no consequence to the grid-generation scheme. In the discrete case, all that is required is that the surface be defined as an ordered set of points describing each cross section of the geometry.

Parallel Hardware Platforms

The grid-generation software was implemented on two MIMD platforms. One of them was the Intel iPSC/860 computer and the other was a network of Sun workstations configured to simulate a MIMD system. Salient features of the two systems are discussed below.

Intel iPSC/860

The Intel iPSC/860 is based on a 64-bit 40-MHz i860 microprocessor. A single computational node of the Langley iPSC/860 system consists of the i860, 8 MB dynamic random access memory, and hardware for communication with other nodes. The system consists of 32 computational nodes arranged in a 5-dimensional hypercube using the direct connect routing module and the hypercube interconnect technology of the earlier 80386-based iPSC/2. The point-to-point aggregate bandwidth is 2.8 MB/sec per channel and the latency for message passing is about 74 μs for message lengths over 100 bytes. Interprocessor communication takes place through the send and receive system calls. Any processor can send a message to any other processor; however, the destination processor does not acquire the message unless it issues a receive. The message passing protocols are implemented with software resulting in high communication overhead.

The complete system is controlled by a system resource module (SRM), which is based on an Intel 80386 processor. This system handles compilation and linking of source programs as well as loading of the executable code into the hypercube nodes and initiating execution.

Network of Sun Workstations

A collection of Sun workstations was used in this study as an alternative parallel computing platform. The Express programming environment allows configuration of a group of networked Sun workstations in order to emulate a multinode parallel computer. Each individual workstation serves as one node of the resulting parallel system. In the present study, a total of eight dissimilar Sun workstations were used. The collection consisted of one SPARCstation 2, two SPARCstation SLC's, and five SPARCstation IPC's. The workstations were connected to each other via Ethernet sockets. Interprocess communications among Express programs running on the network are performed using standard UNIX shared memory and semaphore operations. The three classes of workstations used in this study are characterized by widely different processing speeds.

Programming Models

The grid-generation software was implemented using the node programming model under the native Fortran environment on the iPSC/860. The equivalent model in the Express environment is called the cubix model. This programming model is characterized by the following features:

1. There is no host or master controller program.
2. A single program is written and compiled.
3. This program is loaded into all nodes.
4. The program executes independently on each node.
5. The nodes operate independently on their own data.
6. Nodes share data through message passing.

The most important benefit of this programming model is that the underlying code is essentially the same as if it were executing on a conventional sequential computer. This permits the programmer to utilize usual intuitions when writing, developing, and debugging the code.

Parallel Algorithm and Notes on Implementation

Parallelism

The main grid-generator equations are given by the algebraic relations in equations (15). Once the surface geometries have been specified either analytically or discretely, all parameters can be generated by explicit algebraic formulas. The complete grid-generation procedure is therefore explicit and consequently inherently parallelizable. This inherent parallelism is exploited by devising a strategy for dividing the algorithm into several independent processes to run simultaneously on many processors. The strategy consists of mapping the grid onto a regular computational domain and splitting this domain so that

1. The load on the processors is balanced.
2. Communication among processors is minimized.

For the present algorithm, satisfaction of these requirements is straightforward. To achieve the first goal, the region is subdivided into a number of subregions that equals the number P of processors in the parallel system. The completely explicit nature of the algorithm allows this partitioning to be done in a way that ensures equal amounts of computation in all subregions. The initial surface data are the only relevant data required by the processors. Once these data have been read or generated analytically, the computation of each subregion proceeds independently and requires no interprocessor communication. In the present study, the partitioning strategy was applied at two levels of parallelism, as listed below:

1. At the primary level of parallelism, each planar grid is computed on a different processor of the distributed computing system (fig. 4).
2. At the secondary level of parallelism, each planar grid is broken into several segments. Each segment is a collection of several radial lines of the planar grid. These segments are processed in parallel on different processors (fig. 5).

Partitioning of the grid at levels 1 and 2 can be directly affected by simply distributing relevant loop indices over the processors. The main body of the computation is comprised of a three-level nested loop. Let us denote the loop indices by I, J, and K. The index I is associated with the outermost loop and denotes the number of planar grids in the grid system. The indices J and K are associated with the intermediate and inner loops and they denote the circumferential and radial points, respectively, in a planar grid. Partitioning at level 1 entails distributing the I index in an interleaved fashion over the processors. Level 2, in turn, requires an interleaved distribution of the J index. The particular values of the distributed index of any processor can be easily computed in individual processors as a function of the processor number and the maximum value of the index in the grid system. According to this strategy the code loaded into each processor is essentially identical.

Input/Output

At the end of the computation, the distributed grid segments processed on different processors must be collected in one place to store the ordered aggregate grid system for later use. The usual way of accomplishing this is to have each processor send its portion of the grid to a master node via interprocessor communication. This collection results in substantial communication overhead. In the present study the need for this expensive communication has been obviated by implementing the collection procedure by means of the CFS available on the iPSC/860 computer. The CFS provides the nodes with high-speed simultaneous access to secondary storage. Files reside on a number of disks that connect to the hypercube through SCSI's on I/O nodes. A concurrent file is distributed over the disk drives in blocks ordered in a round-robin fashion.

Parallel read and write statements for the CFS system are available under both the native Intel protocol and the Express protocol. The input geometry for the discretely defined body surface is read into each node using the parallel read operation. For level 1 parallelism, illustrated in figure 4, surface data for sections belonging to each node are read simultaneously. Under the Intel protocol the read operation loads consecutive blocks of data of a specified block size into the nodes in the order of increasing node numbers. This order has the effect of distributing the sections over the processors in an interleaved fashion; hence, there is need for an interleaved distribution of the I index. The Express implementation requires an additional seek operation to determine the starting address of the block to be read into each node. In the case of level 2 parallelism, illustrated in figure 5, all surface points defining one cross section of the geometry are read into each node even though the nodes process only a portion of the sectional grid. This step is done to eliminate interprocessor communication that would be required for transferring adjacent boundary point data for orthogonality calculations.

In the output phase the nodes write their segments of the computed grid as consecutive blocks of equal size. The block size equals the number of points in a sectional grid for level 1 and the number of points in the collection of radial lines in each node for level 2. As in the input phase, the nodes output their blocks in the order of increasing node numbers. Because of the interleaved distribution of loop indices with a stride equal to the number of nodes in the system, the output aggregate grid on CFS preserves the global conceptual I, J, and K ordering of the three-dimensional grid system.

There is an important systemic difference between the Intel and Express implementations of the CFS system. The interface to the CFS used by Express is based on the Intel low level I/O system. The cubix programming model under Express requires that a CFS file be opened on each node to allow simultaneous I/O access. The Intel system, however, has no concept of a file that is open on every node. This difference has been circumvented by implementing the Express interface by opening the file only on node 0 and funneling all data to and from the CFS via node 0. This process has the consequence that CFS operations under Express are somewhat slower than those with the optimized Intel libraries, and some restrictions are placed on which I/O modes can be supported.

Results and Discussion

Description of Test Cases

Two test cases were used to validate the grid-generation methodology and the programming modes. A blended wing-body configuration geometry was used as the test case for the analytic input geometry case. This configuration was defined by 64 cross-sectional planes, each containing 32 points along the circumference of the cross section of the

body surface. Each planar grid for this case contained 49 points along each radial line. A representative body geometry along with two cross-sectional grids is shown in figure 6. A single planar grid for this case is presented in figure 7. Near orthogonality of grid lines and clustering in the vicinity of the body surface are clearly demonstrated in the enlarged view presented in figure 8.

A high-speed transport aircraft was chosen to be the test geometry for the discretely defined input geometry case. This geometry was specified by 64 cross-sectional shapes of the surface geometry, each with 42 points along the circumference. The number of radial grid portions was 49, as in the previous case. A representative grid system for this case is shown in figure 9.

Performance Analysis

A detailed account of the performance of the algorithm on all computing platforms and in all programming modes described above is presented in this section. The execution times reported here are averaged values for several runs in each category. It is important to note that the execution time on the iPSC/860 for an individual case may vary for identical consecutive runs even in the single-user mode. The execution times are affected by a number of subtle factors such as network contention, message timing, caching, and data alignment. It is also important to note that network file transfer traffic to and from the CFS can cause congestion on the communication links in the hypercube because the I/O nodes use the hypercube links to communicate with each other and the service node.

Total execution times in four different categories are presented in tables 1 to 4. These execution times include both computation times and interprocessor communication times. The interprocessor communication times were measured by timing the execution of relevant portions of the code which dealt with passing data between processors. Execution times for the complete grid system in the analytic geometry case are reported with the corresponding number of processors used in table 1. Both the native Intel programming model and the Express programming model display decreasing execution time as the number of processors in the hypercube is increased from 2 to 32. The Express execution times are slightly higher than the corresponding Intel runs because of the communication overhead associated with the Express software. Another factor contributing to the increase in the execution time is the fact that CFS operation is slower with Express, as mentioned previously. Similar trends are seen in the execution times for the discretely defined geometry case presented in table 2. Execution times for the Intel and the Express programming modes in the analytic geometry case are plotted against the number of processors in figure 10. Execution times obtained for a fully vectorized version of the code on a four-processor Cray-2 and an eight-processor Cray Y-MP are also plotted in figure 10. Single-processor execution times for the Cray machines are included for reference. The execution time for the parallel code approaches the Cray execution times as the number of processors increases. An indication of typical interprocessor communication times associated with the algorithm is given in figure 11, wherein communication time has been plotted against the number of processors in the analytic input geometry case. The communication time is shown to be a small fraction of the total execution time. Execution times obtained for level 2 parallelism are presented in tables 3 and 4. At this level, a collection of radial lines belonging to a single planar grid is computed on each processor. The amount of computational activity in each processor is very small in this case, and the total execution time is dominated by I/O activity. The CFS I/O activity does not depend on the number of processors; the result being an insignificant reduction in execution time as the number of processors is increased. Note that the Express execution times are consistently higher than the corresponding times for the Intel programming mode because CFS activity is essentially serialized under Express.

The next series of figures illustrates the variation of the execution time speed-up ratios with the number of processors used. The speed-up ratio S_p is defined as follows:

$$S_p = \frac{T_1}{T_p}$$

where T_p is the execution time of p processors and T_1 is the execution time for a single processor. Speed-up ratios of the analytic input geometry are plotted against the number of processors in figure 12. Results for both the Intel programming mode and the Express programming mode are presented. These results are for level 1 parallelism (i.e., for the complete three-dimensional grid system). Both programming modes result in good correlation of the speed-up ratio with the number of processors used. Ideally the speed-up ratio S_p for P processors should be equal to p; however, in a realistic application, the presence of nonscalable factors such as overhead, broadcasting of initial run-control parameters to all processors, and certain I/O operations will lower the value of S_p somewhat. The speed-up-ratio curves in figure 12 are nearly linear an indication of consistent

reductions in execution time as the number of processors is increased. The speed-up ratios in the Express mode are lower than the corresponding ratios for the Intel programming mode because of overhead and slower CFS operations associated with the Express environment. Speed-up ratios plotted against the number of processors in the discrete geometry case are presented in figure 13. Similar variations in speed-up ratios are again noted in both the Intel mode and the Express mode.

Performance results obtained by running the code under the Express environment on a network of eight Sun workstations are presented in the next two plots. Figures 14 and 15 illustrate the performance of the algorithm while computing the complete three-dimensional grid system in the analytic and discrete input geometry cases, respectively. Speed-up ratios increased with increasing numbers of processors. However, it is important to remember that the workstations in the network varied widely in their computing power and characteristics, and consequently, a consistent correlation to performance is not expected.

Conclusions

An algebraic grid-generation method based on homotopic relations has been demonstrated to be highly parallelizable at several levels of domain decomposition. Performance of the algorithm on the Intel iPSC/860 machine with 32 nodes has displayed consistent correlation of speed-up ratios with the number of processors. A strength of the algorithm is its relatively low need for interprocessor communications. The Express algorithm was also executed successfully on a network of Sun workstations to validate the possibility of running parallel codes without a true parallel machine. The consistency of speed-up ratios correlating with the number of processors on a network of workstations is expected to improve with uniformity of the computing characteristics of the individual workstations.

NASA Langley Research Center
Hampton, VA 23681-0001
December 3, 1993

References

1. Holst, Terry L.; Salas, Manuel D.; and Claus, Russell W.: The NASA Computational Aerosciences Program—Toward TeraFLOPS Computing. AIAA Paper No. 92-0558, Jan. 1992.

2. Gentzsch, W.; and Häuser, J.: Mesh Generation on Parallel Computers. *Numerical Grid Generation in Computational Fluid Mechanics '88*, S. Sengupta, J. Häuser, P. R. Eiseman, and J. F. Thompson, eds., Pineridge Press, Ltd. (Swansea, Wales), 1988, pp. 113–123.

3. Berger, Marsha J.: Adaptive Mesh Refinement for Parallel Processors. *Parallel Processing for Scientific Computing*, Soc. for Industrial and Applied Mathematics, 1989, pp. 182–194.

4. Löhner, Rainald; Camberos, Jose; and Merriam, Marshal: Parallel Unstructured Grid Generation. *A Collection of Technical Papers—Tenth AIAA Computational Fluid Dynamics Conference*, June 1991, pp. 627–637. (Available as AIAA-91-1582.)

5. Moitra, Anutosh: *An Algebraic Homotopy Method for Generating Quasi-Three-Dimensional Grids for High-Speed Configurations.* NASA CR-4242, 1989.

6. Moitra, Anutosh: *HOMAR: A Computer Code for Generating Homotopic Grids Using Algebraic Relations—Users' Manual.* NASA CR-4243, 1989.

Table 1. Execution Time for Complete Grid in Analytic Geometry Case

Number of processors	Execution time, sec		Speed-up ratio	
	iPSC/860	Express-iPSC/860	iPSC/860	Express-iPSC/860
1	65.12	69.91	1.00	1.00
2	32.89	35.31	1.98	1.98
4	16.54	17.70	3.93	3.94
8	8.43	9.50	7.72	7.36
16	4.55	5.58	14.31	12.52
32	2.81	3.70	23.17	18.89

Table 2. Execution Time for Complete Grid in Discrete Geometry Case

Number of processors	Execution time, sec		Speed-up ratio	
	iPSC/860	Express-iPSC/860	iPSC/860	Express-iPSC/860
1	119.63	121.22	1.00	1.00
2	60.73	63.80	1.97	1.90
4	30.91	33.60	3.87	3.60
8	16.68	17.99	7.63	6.74
16	8.44	10.38	14.17	11.68
32	4.87	6.99	24.56	17.35

Table 3. Execution Time for Single Planar Grid in Analytic Geometry Case

Number of processors	Execution time, sec	
	iPSC/860	Express-iPSC/860[a]
2	0.766	2.56
4	.514	2.72
8	.412	2.81
16	.381	3.02

[a]Express serializes CFS I/O operations.

Table 4. Execution Time for Single Planar Grid in Discrete Geometry Case

Number of processors	Execution time, sec	
	iPSC/860	Express-iPSC/860[a]
2	1.200	3.19
4	.835	3.99
8	.659	4.40
16	.630	4.54
32	.582	5.39

[a]Express serializes CFS I/O operations.

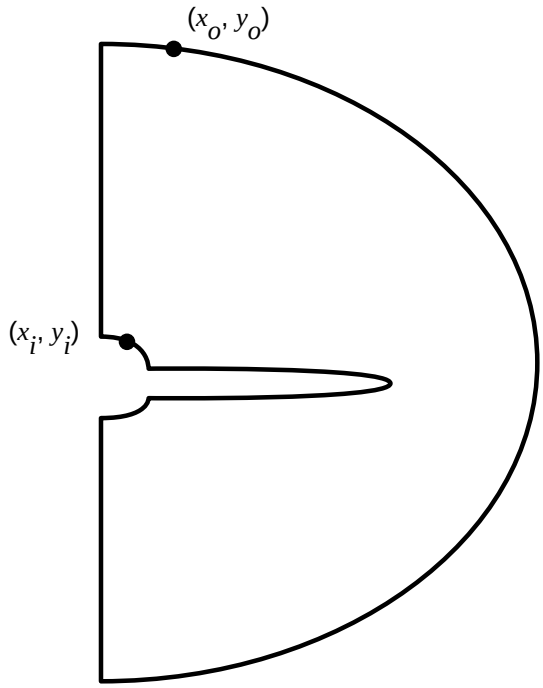


Figure 1. Corresponding inner and outer boundary points.

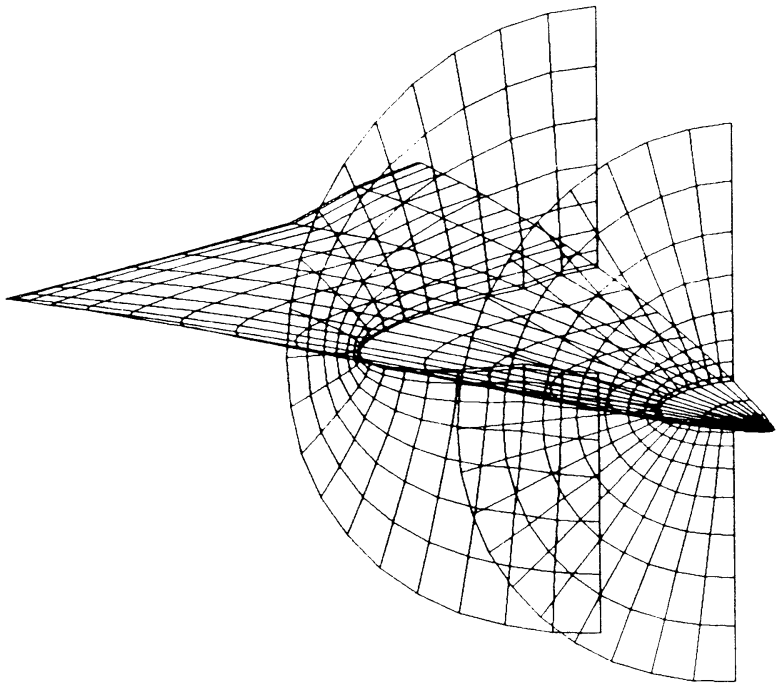


Figure 2. Schematic representation of quasi-three-dimensional grid system.

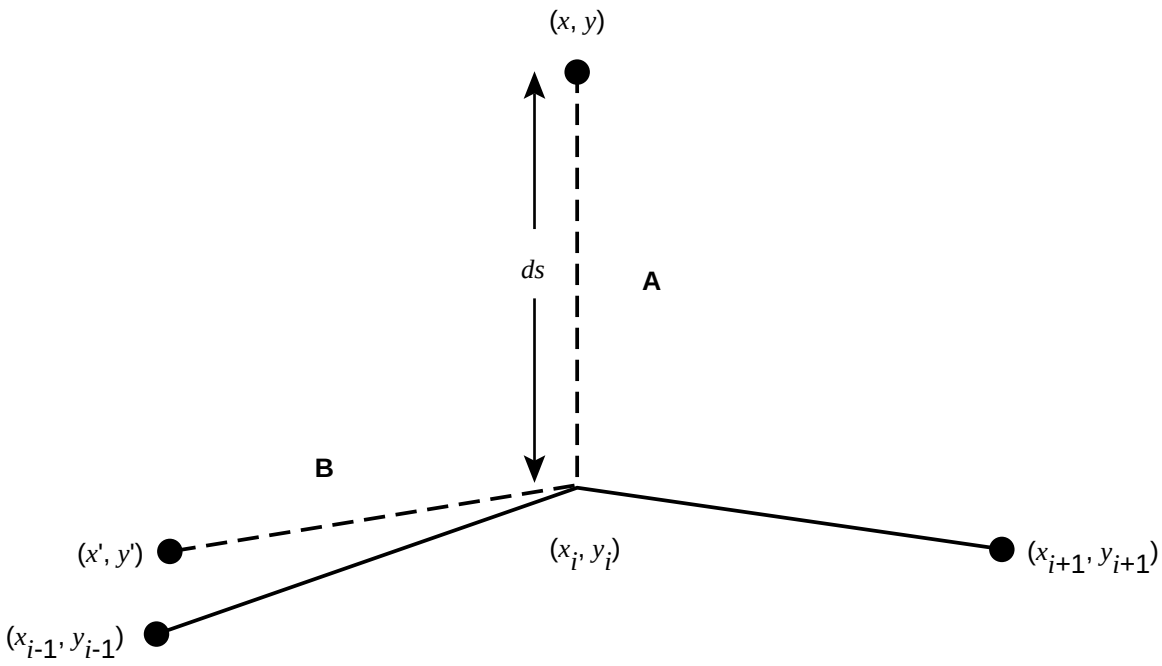


Figure 3. Vectors used in grid orthogonalization.

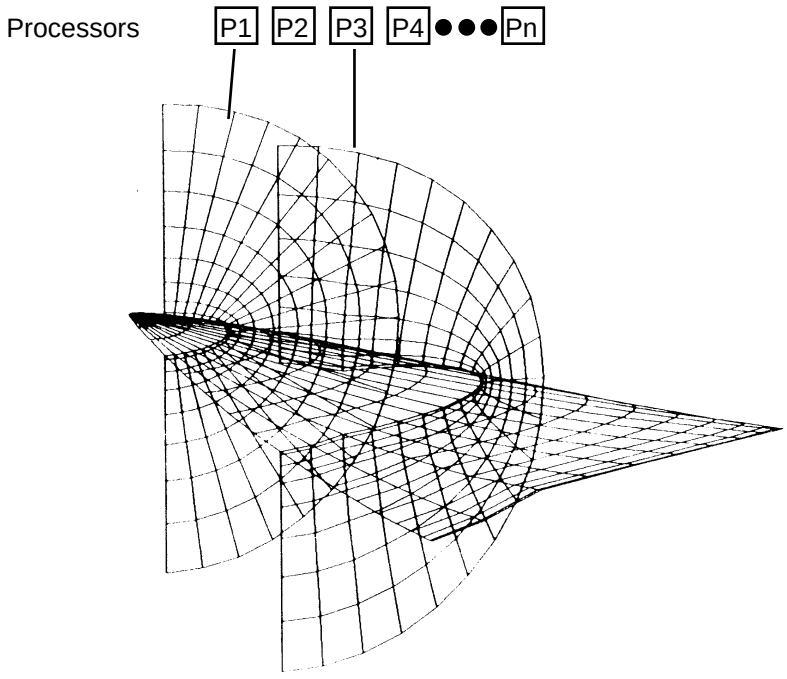


Figure 4. Schematic of parallelism at primary level.

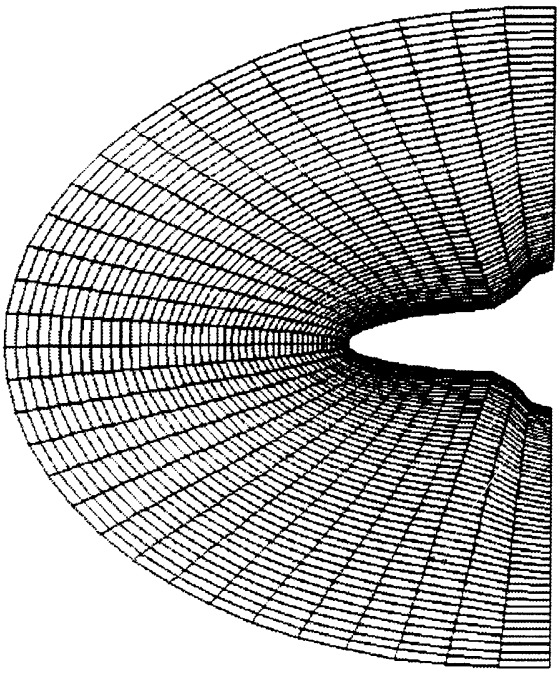


Figure 7. Representative computed planar grid for analytic case.

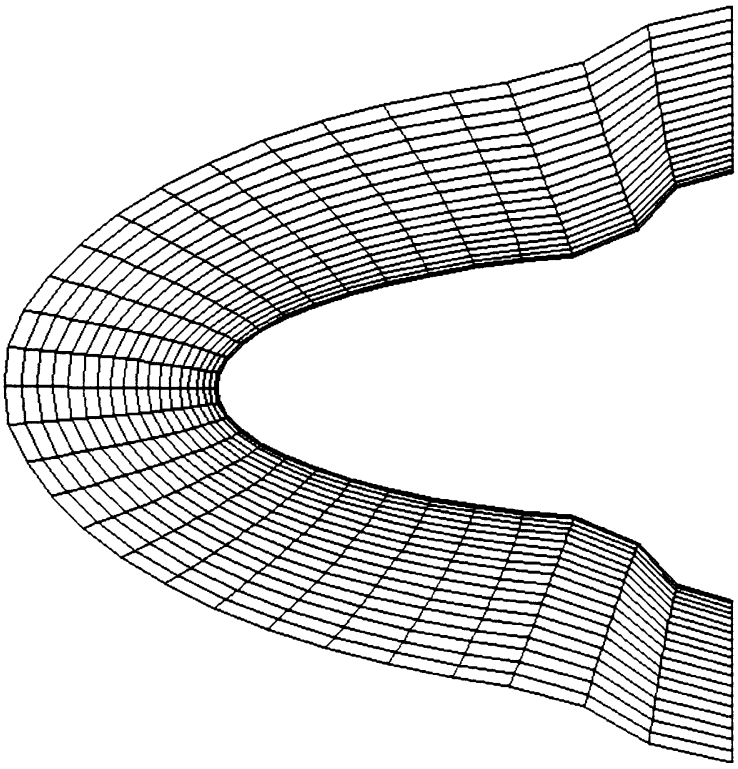


Figure 8. Enlarged view of representative computed planar grid.

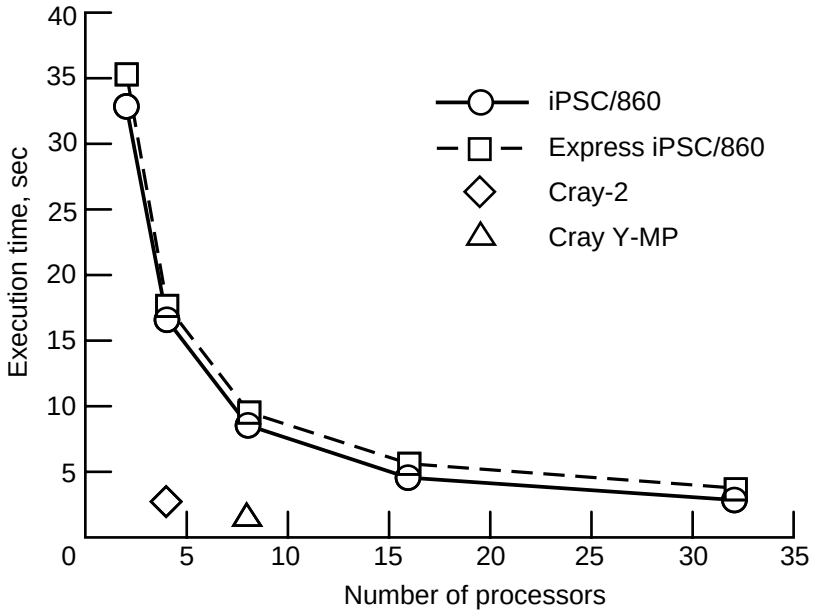


Figure 10. Variation of total execution time with number of processors.

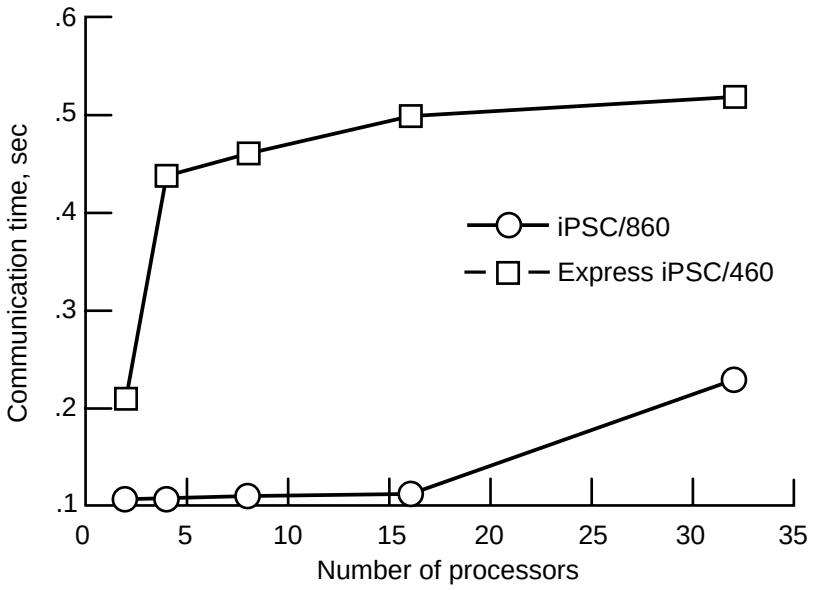


Figure 11. Variation of communication time with number of processors.

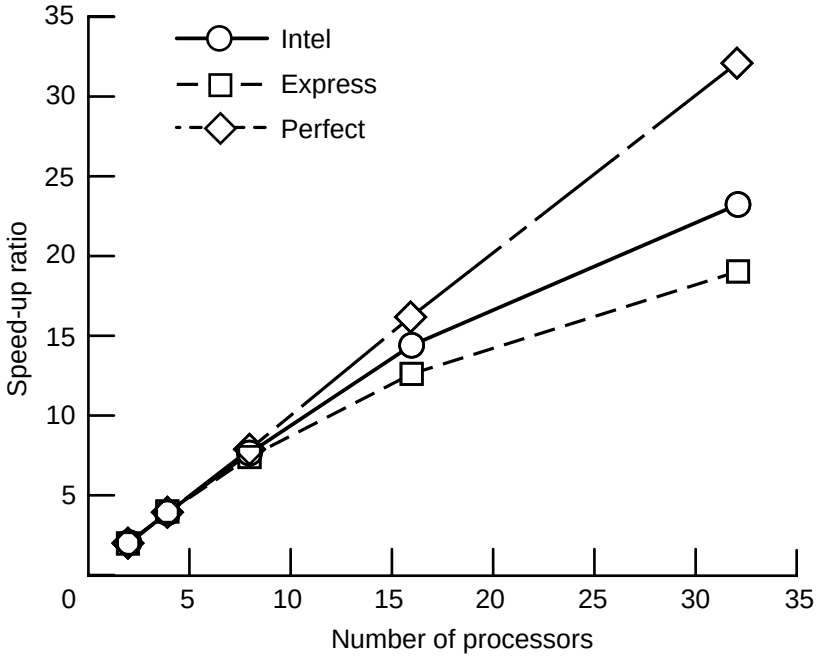


Figure 12. Variation of speed-up ratio with number of processors for analytic geometry case on iPSC/860 machine.

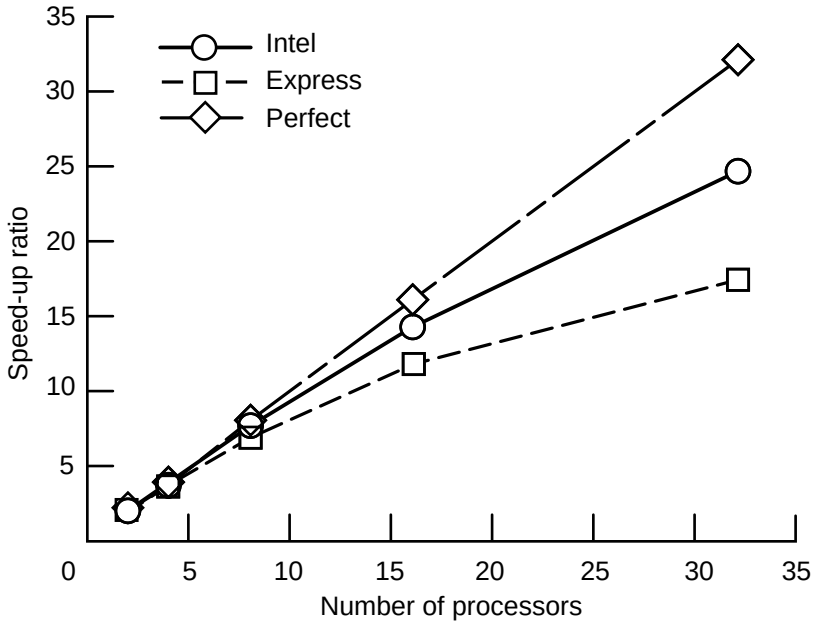


Figure 13. Variation of speed-up ratio with number of processors for discrete geometry case on iPSC/860 machine.

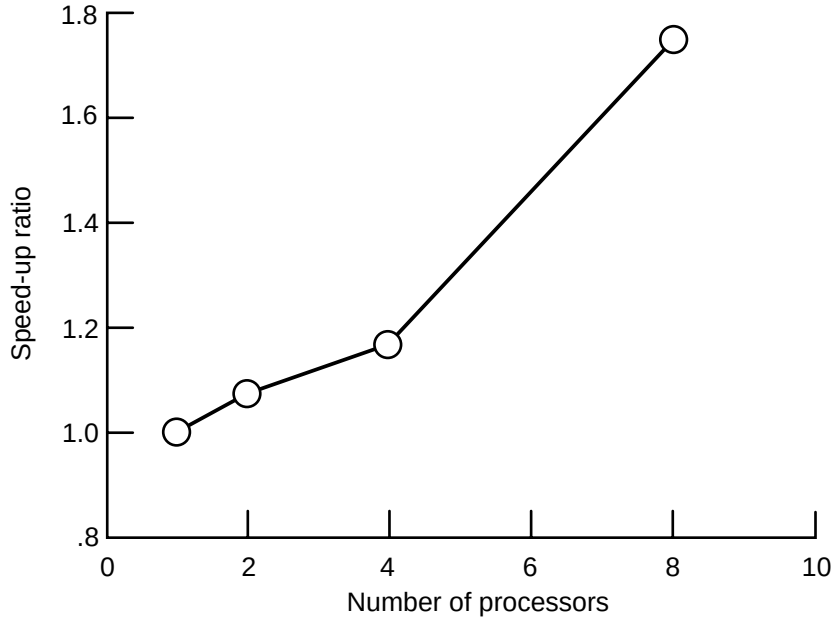


Figure 14. Variation of speed-up ratio with number of processors for analytic geometry case on Sun network.

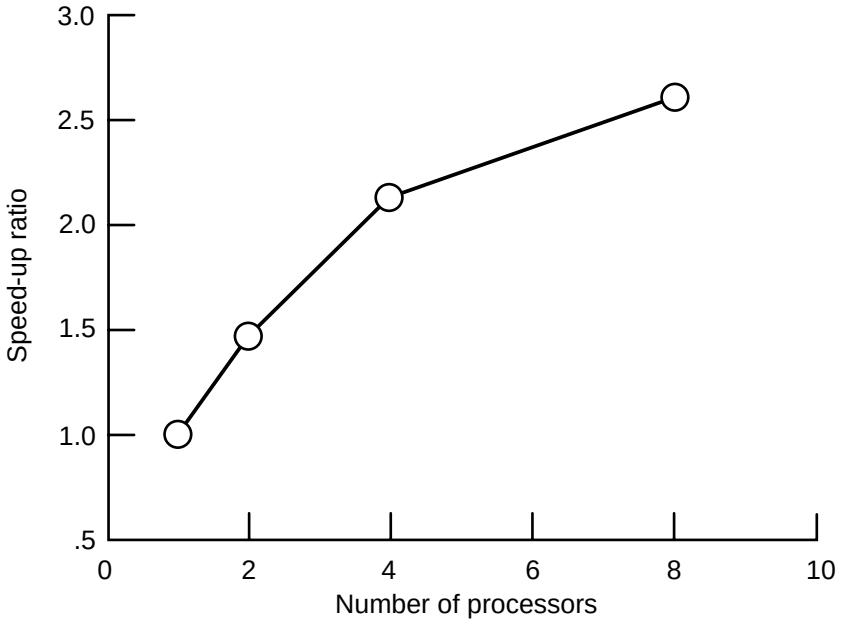


Figure 15. Variation of speed-up ratio with number of processors for discrete geometry case on Sun network.

Figure 5. Schematic of parallelism at secondary level.

Figure 6. Representative computed grid for analytic case.

Figure 9. Representative computed grid for discrete case.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1994	Technical Paper

4. TITLE AND SUBTITLE
Parallel Grid Generation Algorithm for Distributed Memory Computers

5. FUNDING NUMBERS
WU 505-90-53-02

6. AUTHOR(S)
Stuti Moitra and Anutosh Moitra

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17249

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TP-3429

11. SUPPLEMENTARY NOTES
Stuti Moitra: Langley Research Center, Hampton, VA; and Anutosh Moitra: High Technology Corporation, Hampton, VA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
A parallel grid-generation algorithm and its implementation on the Intel iPSC/860 computer are described. The grid-generation scheme is based on an algebraic formulation of homotopic relations. Methods for utilizing the inherent parallelism of the grid-generation scheme are described, and implementation of multiple levels of parallelism on multiple instruction multiple data machines are indicated. The algorithm is capable of providing near orthogonality and spacing control at solid boundaries while requiring minimal interprocessor communications. Results obtained on the Intel hypercube for a blended wing-body configuration are used to demonstrate the effectiveness of the algorithm. Fortran implementations based on the native programming model of the iPSC/860 computer and the Express system of software tools are reported. Computational gains in execution time speed-up ratios are given.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Grid generation; Parallel algorithm	20
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		